EPISODE 15

[INTERVIEW]

[0:00:00.3] JM: Kent Beck, welcome to Software Engineering Daily.

[0:00:02.6] KB: Thank you very much, Jeff. It's a pleasure to be here.

[0:00:05.5] JM: You joined Facebook in 2011. What did you initially do with the company?

[0:00:11.8] KB: Toward the end of 2010, I got a call from a recruiter at this company Facebook, which was not – it was it was known to be very successful, but it wasn't a thing, especially in the engineering world. I went and interviewed. I saw an engineering culture that was very, very different than anything that I had seen before.

My last interview of the day was with Mike Schroepfer and I said, "This culture really is unique, but maintaining it and the evolving it is going to be hard work. I would like to come and do that." My initial pitch to the company was as a culture carrier, amplifier, observer, commenter, tweaker.

[0:01:16.9] JM: What makes you say that the Facebook culture as of 2011 would be hard to scale?

[0:01:25.5] KB: I didn't know that it would be – Okay. What was clearly going to be hard to scale was avoiding mean reversion. It was 2,000 employees total, 700 engineers. It was obviously going to grow very quickly as these things do in the valley. A bunch of the people they were going to hire in would be coming in from the Googles and Microsofts of the world and would bring a very different set of values and a very different set of practices.

The natural tendency would be for Facebook engineering culture to just revert to the least common denominator, which everybody seems – that's the attractor. Everybody falls back into lots of planning and long cycles and more handoffs and bigger batches and all the stuff that makes short-term small-scale sense and that kills long-term effectiveness.

[0:02:39.3] JM: Talk about that in more detail. It sounds like you have a perspective that there's some overly processed, laden engineering structure that is endemic to companies as they get big.

[0:02:56.9] KB: Not even as they get big. Sometimes they start out that way, which really astonishes me, like why would you put the shackles on before you dove into the pool the first time? That doesn't make any sense. The unique – every software process has to be shaped by context. Facebook's context was in some ways, absolutely unique, because you had this scaling component that was just insane, trying to make computers do things that they had never done before, in terms of the number of people and the amount of data and so on.

At the same time, they were still exploring the space of what social media interaction looked like. They were going to simultaneously have to solve these difficult engineering problems and continue to efficiently explore this gigantic space of all the possible social interactions you can mediate with a computer.

That was the context. I think every startup deals with that first part, where you're exploring some space and you need to find a niche that is uniquely yours. You have to find that engine of growth. Efficiently exploring that space is a very different task than making incremental improvements. People fall back on the tactics, strategies and value systems that are suited for incremental improvements.

Okay, so let me take a step back. I see this place, it's crazy, it looks like a clown show and yet, things are working really well. They weren't doing the things in my books. I like to joke, I don't mind if people don't do this stuff in my books. I just want them to fail. They weren't doing that. I thought, well – My first thought is I'll come in and explain how this stuff works.

In the back of my mind, there is this mystery of this bumble bee. In theory, this process should be a disaster and in practice, it's working extremely well at two things at the same time; at scaling and at exploration. I wanted to figure that out. The first hackathon, you could sign up to give classes. This was during boot camp. I hope some of these explain boot camp.

[0:05:58.6] JM: Yes, they have.

[0:05:59.5] KB: Okay. I thought, well I'll give a class on test-driven development, because here I am and I can – nobody's using TBD, so well, of course, they'll want to learn from me. In the signup sheet, I went and checked just before I left. I went and checked the old, old wiki and the evidence is still there.

The class just before mine on the list of classes was about advanced techniques in Excel and it was full and there was a waitlist. The class just after mine was on Argentinian tango and it was full and there was a waitlist. I had zero people sign up for my class. I thought, "How am I going to have any impact here, if people don't listen to me?"

I deliberately chose to forget everything I knew about software engineering. I just said, "I'm going to try and be a programmer and I'm going to watch what people do. I'm just going to copy what they do. If somebody says this is too diff since that one, it will be too diffs. If somebody says you need tests for this, I'll write tests. If they say you don't need tests for that. Why are you writing tests? Then I won't write tests, even if I think that's my – that's the natural thing to do.

That's the only way one, that I was going to be able to explore this mystery of how this software engineering process worked. Two, is the only way that I was going to have any influence, because clearly, nobody was going to listen to me based on reputation.

[0:07:38.0] JM: The tension that you're describing, this arises from the tension between maintenance, the maintaining of an existing software product that's working and exploring new ideas. The tension manifests, because when you're a business like Facebook and you develop a news feed product and a messenger product and an ads product, and all these things are having a lot of success, there are natural pressures to put a bunch of engineers on the maintenance of those products and the expansion of those, the gradual slow iterative expansion of those products.

You should do that. You should maintain them. You should improve them. At the same time, when you have such a greenfield opportunity like Facebook, the greenfield opportunity being social interaction on the internet mediated by real identity, you do want to continue to explore those new ideas. How were those tensions being resolved? As you solved the mystery over your seven years there, what was Facebook doing to resolve that tension?

[0:08:54.6] KB: Sure. It took five years to parse it out. I spent the first, almost a year working on privacy and on messaging back-end, and discovered that I was probably the worst C++ programmer at Facebook. I got a bad review. It was clear that just trying to sling code was not my differential advantage there. I started coaching. I had done a fair amount of one-on-one coaching with engineers before. There were no coaches at Facebook. I could see that there were engineers with tons of unrealized potential.

Because Facebook was solving unprecedented problems, there was no way they could hire somebody to solve them. They had a big bunch of the technical horsepower had to be generated in-house. I started this coaching program called good to great and began working with engineers one on one. Ended up coaching maybe a 150 or 200 engineers.

Personally, the program matched up other senior engineers with junior engineers who got more coaching. My students were demonstrably faster at getting promotions. They were twice as likely to get promoted in the year following coaching than their peers who didn't get coached, all other things being as much equal as possible.

That was working, although it was odd to be – I was an insider, but I was also an outsider, right? Because I wasn't slinging a whole bunch of code. I was working with people and helping them do their jobs better. That gave me the the ability to work with engineers all over the codebase, all over the organization, people in brand new stuff, people refining older and more established stuff, people in infrastructure. I got to see all over the codebase.

As I said, took me five years before I realized, “Okay, here's the structure that makes all this go.” This was not explicit, but the behavior matches very carefully and it goes like this, that there are – there's a style of project that's really good at exploring four new engines of growth.

You have very little to lose in that moment. This is where move fast and break things is exactly the right attitude to take. If you have nothing to lose, breaking something has no cost, so just go for it. I call that the exploration phase. You want to place as many bets as possible, because you can't analyze your way to success.

Successful exploration is always a surprise, but it's not random. You're actively searching for this thing. Then holy cow, like the Facebook live video was something that just took off like a rocket. As soon as something takes off, the game changes. All the trade-offs change. Now you're in this vertical growth phase. That vertical growth phase has its own set of rules, its own set of trade-offs. I call that the expand phase. Its own style of project management, that's where war rooms make a lot of sense, get everybody sitting together. It's a relatively brief period, because you have to overcome one hurdle to growth after another after another after another, is very intense. That's the place where extra – that's the only place on this curve that extra hours actually makes any difference.

You come out of that expansion exhausted and you go into what we would call the growth teams, the extract part of the curve, where again, all the tradeoffs change. Now you're at massive scale. While you're expanding, throwing money at problems makes perfect sense, because you're at risk of failing entirely if you don't overcome some barrier.

When you're extracting, now you've got tens of thousands, hundreds of thousands of servers. Reducing the number of servers by 10% is a hugely profitable project for one programmer. While you're expanding, reducing the number of servers by 10%, if you go from 20 to 18, who cares? If you're doing that, you're not doing something else to actually scale it.

What I noticed is informally, projects in those three phases were managed completely differently. A couple of people would peel off and just start trying stuff in some area that they were interested in. That's the explore phase. Something they did would take off and start growing quickly, and then the infrastructure would fall over because it was a new – some new demand. People would join the project who didn't like trying out new things, but loved diving into hairy, technical, unprecedented technical problems.

Steve Grimm was an early, early Facebook engineer, is the first person that I met when I walked through the door at Facebook. I talked with him about this and he said, yeah. He just waited for the chance to have another unprecedented technical problem. That expand phase was staffed differently, managed differently, funding wasn't – that's its own whole topic, but it was treated very differently. This would manifest, you would see a team go in for a Zuck review and come out bubbling and happy and ready to do the next thing.

A month later, they would go for a review and report that they'd been doing the same stuff for another month since that was working and come out just looking like their dog died, because they've gotten reamed. Why are you still trying out all this stuff? This thing is working and its growth is stalling. Focus in on that. Well, it seemed capricious at first, but if you – once I could step back and say, “Oh, the tradeoff change. You've got this exploration phase, the expansion phase. If you don't notice that you've made the difference and you keep exploring and trying out this and that, you miss the opportunity to expand.” Underneath as an unspoken structure, this explore, expand, extract was going on all the time.

[0:16:44.0] JM: Kent, real quick. Occasionally, I'm hearing your mic brush against either your shirt or your beard. If you don't mind, just try to try to position yourself so that it's hanging in a way where it won't brush against anything.

[0:16:56.1] KB: Okay, will do.

[0:16:59.7] JM: This culture of engineering at Facebook, is there something unique about Facebook that allows that culture to work for the Facebook set of products, or would this work for any company, for any set of products?

[0:17:23.7] KB: Facebook 2017 was a very different beast than, or 2018 when I left, than 2011 when I joined. I think the things – this balance between exploring and extracting as you said, that's a universal. Once you've gone through that curve once, the natural tendency of everybody is to treat all projects, like extract projects. The really successful companies feed back into more explorer projects and keep treating them as a special – as a different project.

KPIs for example, is a great thing to have in extract. You know what the levers are, you know if you increase this by 4%, that goes down by 6%, or whatever. That's fine. KPIs in explorer don't make any sense, whatsoever. I talked to an ads manager who is really frustrated. Facebook is very metrics-oriented company. He said, “Either my projects have shows zero improvement, or a thousand X of what our goal was.” That's very characteristic of explore projects. It's really binary.

If somebody says to you, “Well, we'll get a 20% ROI on this completely speculative project,” they're just lying. Either this is going to give you zero, or it's going to be spectacular and then you have to move into this expand style to find out how spectacular. Did that answer your question?

[0:19:22.8] JM: I think so. Let's go back to that depiction of the team that builds a new feature. They go into a Zuck review. The first time they go into the Zuck review, the review is fantastic, they've built something new, it's gaining traction. The second time they come into a Zuck review, maybe a month later, they've been doing the same thing, it's working, but the review is not as positive. Help me understand. What exactly are you referring to there? Is there some adjustment that a team needs to make when a product is working? Do they need to put it on maintenance mode? Do they need to start trying other new things? What's going on in that shift?

[0:20:20.0] KB: I see. No, they don't need to put it into maintenance mode. No, they don't need to explore more. What they need to do is to take live videos as an example. It launched. It was far more successful than people expected and it put new strains on the infrastructure. I mean, at that point Facebook had fantastic infrastructure. Even at that live video, was not – the latency was too long, the number of successful connections, the percentage of successful connections was too low. Yet, there were people on the team who wanted – okay, now who wanted to continue exploring?

I had a conversation with folks on the team. I said, “Hang on, the game has changed. This is not about finding the next feature that might drive growth. You already have the feature that's driving growth, but the infrastructure is impeding that growth.” You need to say, “Yes, we will get to all those cool exploratory features when the time comes. Right now, we just have to make what we're doing work.” If that means cutting features, or artificially reducing the demand for the product, in order to get over the next scaling barrier, then then that's what we'll do.

Another analogy that I use is it's as if you're playing soccer and then the referee blows the whistle and out comes this oblong ball and your opponent's trot off the field and this new opposing team comes on and now you're playing rugby. Well, if you keep by soccer rules, you're just going to get crushed. You have to adapt. That was the moment.

The tension was hey, we like trying crazy new features and launching a new feature every week. When you hit vertical growth, you have to attend to the vertical growth. That doesn't happen very often. That exploratory phase is profitable. You have a small chance of a big payoff. The extract phase is profitable, because you have a large chance of a relatively small percentage-wise, payoff. Expand is its own thing. That's the thing that I learned at Facebook was there is a particular style. It's the only part of the product development cycle where you have a large chance of a large payoff. Punting an expansion phase is really, really expensive.

[0:23:19.6] JM: One of the motivations for doing this series of interviews is to contrast the Facebook engineering culture to that of Google. Because there's a sense that the Facebook engineering culture was largely a replica of what Google did successfully, or what Microsoft did successfully. It's clearly its own distinct engineering culture. How does Facebook engineering culture contrast with the previous successful tech giants?

[0:24:02.7] KB: I spent most of the aughts on a goat farm in southern Oregon. I can't give you the insider view of that. I think the extreme programming was an attempt to get away from big upfront planning, or to put it positively, to move decisions to where they could most effectively be made, instead of piling them up in big batches either too early or too late.

One thing is there's a myth about Google culture that Jeff Dean sits in a dark room and thinks really hard and then comes up with some amazing piece of infrastructure. I read the those early Google infrastructure papers with a lot of interest. I think there was a literally a paragraph in there that really caught my interest, which was we wrote three web crawlers. As soon as we finished one, it would start to show signs that it was going to fall over. We hacked together another one and another one. Then finally, said, we have to solve these problems once and for all.

It’s I don't believe that even Google, like the story is oh, somebody just thinks really hard and then comes up with some big piece of infra. I don't believe it. There's a lot of iteration. There's a lot of stuff that fails and gets replaced quickly. The folks that I talk to now that are trying to create a culture from scratch, that mythology of well, think really hard about your infrastructure, because you don't want to be in a position where you want to scale and you can't scale.

Well, at what cost? If that means that you're going to explore that much more slowly, you're just putting a parking brake on your product and your company. You're trying to drive and the brakes are on, because you're slowing down.

At Facebook, it was very much explicitly – There was a poster of Facebook that said, “Nothing at Facebook is somebody else's problem.” That's the foundation of Facebook engineering culture. If you saw something, you wouldn't necessarily just go fix it, because you can't solve all problems. If you didn't, it's because you chose not to. The way that played out in explore projects is if you needed a key value store for your – and this is an actual example. If you needed a key value store, you would just write one of your own, or you'd use one that was open source or something.

At one point, there were four key value stores in production, at scale, at the same time. You might say, “Well, that's inefficient.” No, that enables efficient exploration. If all those teams were told, “No, you have to wait for the corporate standard key value store to come out, the explorations wouldn't have happened, the expansions wouldn't have happened and you wouldn't even get to the extract stage.”

Now you got four key value stores and they all have roughly the same API. An infrastructure project to go from four to three is hugely profitable, right? You put four engineers on, you save 2,000 servers. That's super. You go from three to two and two to one. By the time you've gone through that process, you have one really awesome key value store. At the same time, you've been able to efficiently explore this gigantic and very valuable space of what all features could we be adding?

[0:28:27.9] JM: That reminds me of a question that I've – I haven't quite understood yet, which is why Facebook doesn't use public cloud infrastructure at all. I understand that Facebook was started, I think it went 2005 or something, but before the cloud was a thing. The cloud got started in 06, or 2007, somewhere around then. It didn't really reach maturity until probably, I don't know, 2011 or 2012. By then, Facebook had really well-developed infrastructure.

Even then, I would assume that there are useful cloud technologies to take off the shelf and use as APIs, or database-as-a-service. If Facebook was so open to exploring new technologies and picking up random key value stores, why haven't they adopted any cloud technologies?

[0:29:22.2] KB: Because nothing works at Facebook scale. Nothing off-the-shelf works at Facebook scale. If you buy networking equipment, Facebook is going to put that equipment through hell in a way that nobody else does. That's a consequence of working at this unprecedented scale. It wouldn't make sense for a vendor to make a product good enough for Facebook, because they only have – they would only have three customers in the world. It's much better for them to have a product that's not quite as capable and they can sell to thousands of people.

[0:30:07.9] JM: How did Facebook – I mean, I'm not sure if you're the person asked, but since you had a tour of the company, how did Facebook adapt to unprecedented scale?

[0:30:23.5] KB: Well, in the expand phase, in these vertical growth phases where you're hitting unprecedented barriers to growth, they had the world's leading technical expert available to jump in. This is another part of Facebook engineering culture that may be a little subtle. That person was working on some extract project that was very profitable.

You're tuning some database, say, or improving some network equipment, or something like that. That project they're working on is very profitable and they – but that engineer hears – overhear somebody talking about whatever's failing about the hot new feature. It's their responsibility, because nothing at Facebook is somebody else's responsibility, is their responsibility say, “Oh, I would like to work on that.” That engineer then says to their manager, “Hey, I'm going to go work on this. It's growing quickly.”

It's their manager's job then, now the – your leading technical expert just told you that they're off the project to work on this thing, which hasn't even achieved scale yet. It's just growing fast. It's that manager’s job to say, “Okay, good. Bring in the next person and to take over whatever important responsibilities that expander just left behind, in order to jump on the next piece of scaling that otherwise wouldn't happen.” Did that answer your question?

[0:32:15.1] JM: Yeah, it does. When you joined Facebook, my understanding is that around that time, Facebook really didn't have much testing. It's ironic, because you were the creator of extreme programming. It was highly dependent on the process of writing unit tests and then writing the features. Facebook clearly reversed that process and was able to be successful, despite the fact that they wrote their features before they wrote their tests.

You often hear that you need unit tests in order to build things, like continuous integration. Continuous integration lets you move much faster, because you have this battery of unit tests that runs at every new build. I don't think Facebook had that stuff until later. How was Facebook able to move so fast with such a low amount of unit testing?

[0:33:21.4] KB: Yes. That was part of the puzzle when I arrived. One of those hey, this isn't what in the books and yet, it seems to be working. The answer that I came to is that while – There's a couple parts of it. One is, how many of your problems can you test for and how many problems only show up in production? If problems – if you're writing a simple calculation, but it might not scale in production, you can't write unit tests for it.

The Facebook answer is don't. That's part of it is depending on the ratio of how many problems is it possible to test for before production, versus after, if that ratio is skewed towards hey, stuff only fails in production, then don't write any tests.

The second part of the answer is tests are a form of feedback and Facebook engineers had many, many other forms of feedback. You'd write on your development server and you'd try stuff out. Then it would go through code review. That's a second form of feedback. Then it would – at that point, it would roll to the internal site, so you get more feedback from that. Then it would go through the deployment process, where it would get rolled out to a small number of machines and then more and more and you get feedback from that.

Then logging and operational awareness was just part of the engineering culture, so you'd get feedback post-production of how your feature actually was behaving. There's a bunch of feedback loops in place. Unit tests occupied for most development of Facebook, just the cost benefit, the amount of feedback they added and the timeliness of the that feedback and the most of achieving that feedback just wasn't worth it. In boot camp, you're supposed to put code into production the first week. I was very careful to write tests and do everything properly.

I got in a fair amount of heat, because my first feature didn't land for three weeks. People are like, “Man, I don't know how this is going to work out.” Well, and I was wondering that too. I had a huge case of imposter syndrome when I landed at Facebook and realized just how different everything was.

Then the tests that I had written broke almost immediately. They were deleted. That was one of the things that surprised me. If you had a test and it failed, but the site was up, they just delete the test. If you had tests that were intermittent, that were non-deterministic, they were just deleted. At first, I was shocked. Like, delete a test. This is producing noise and it's not producing signal. If you eliminate this noise production, per definition the situation is clearer all of a sudden. The fact that you wish that you had a test for something, well you didn't. Yeah, just chuck it and let's move on.

[0:37:03.7] JM: Now you probably wouldn't want to take that approach to nuclear power plant software, or electricity grid software. That seems like a practice that is uniquely useful for early Facebook, where this thing wasn't yet a communications utility. It was more of a fun thing to do akin to television.

[0:37:32.0] KB: Another lesson I learned at Facebook is the difference between reversible and irreversible decisions. If you roll out a feature in New Zealand say, and people don't like it and you can easily turn it off, then that’s a reversible decision. Facebook engineering spend a lot of effort on making decisions reversible. Sometimes you have to do “extra engineering” to make a decision reversible.

I talked with one of the managers and the network infrastructure and he said, when they got a new piece of networking gear in and it went to the characterization labs just to put it through its paces, they mostly didn't care about its performance. What they cared about was can we turn this thing off safely? That's an investment in reversibility, all the feature flag stuff.

That was one of the lessons I had to learn. People in code reviews would say, “Oh, you definitely need to put this behind a feature flag. I’m like, “What? Why?” My attitude is let me test it, make sure that it works and then I'll just put it out.

The hard learned wisdom there was that there's a bunch of stuff you can't test for. Yes, you have to introduce additional complexity to put the flags in, but if you don't, you're going to roll something out and your only recourse is going to be hotfix rollout, which makes the release engineering grumpy and you did not want to make release engineering grumpy.

[0:39:27.5] JM: What's the role of a manager at Facebook?

[0:39:34.8] KB: That changed a fair amount in the time I was there. When I got there, engineers at all levels were expected to exercise a lot of initiative. There's a beautiful article by Yu Sheng Wang about Facebook's early engineering management philosophy. I can't recall the title of it. Maybe we can find it later. The upshot is Facebook deliberately chose to make engineering easier and the engineering management harder. If there was ever a tradeoff, they would make the manager’s life harder and the engineer’s life and job easier. Now, how do we get on this question?

[0:40:28.6] JM: Well, I asked you what's the role of a manager in a modern – in Facebook?

[0:40:33.7] KB: Yeah, okay. Managers had to – the first role is to attract talent. A manager got a headcount allocation for their team, but that was really loosey-goosey in the early days. Just because you had allocation, didn't mean you had engineers. You had to sell your project.

If you couldn't, then you didn't get any engineers and your project just didn't go anywhere. You had to attract people. Nobody was going to force an engineer to work on your team, so the managers who had bad teams or they were bad managers, people would just go do something else. Managers tend to get weeded out quickly in a way that I didn't expect to see.

A certain amount of arranging for technical collaboration, again it was on the engineers to organize their work together for the most part. Different managers were different, some were more hands-on technically and some less so. To negotiate for headcount allocation and to – and career encouragement for engineers. Advocating for engineers in the performance review process.

[0:42:24.1] JM: What distinguishes Mark Zuckerberg as a leader?

[0:42:27.0] KB: I did not miss a weekly Q&A for the first four or five years, because something new for me would come out every week. It's a combination of – In fact, I never met him face to face in seven years there. I don't have any of those good stories. This implicit understanding that projects go through these three phases and a taste for when to shift gears. When is something a blip and when is something a genuine evidence that this is worth expanding? Sometimes it's dead obvious and sometimes it isn't.

Another is a willingness to abandon metrics that have served their purpose. That was a big surprise to me. Okay, time to interaction is good enough now, don't let it slip, but that's not the priority anymore. I was used to the Monty Python style, where the most important metric is A and B, the most important metrics are A, B and C, right? It would just get piled on and piled on. To see him stand up and say, "Yes, we've harped on this issue and we're not going to do that anymore," I thought was really remarkable.

He's also very good at attracting very good people. Somebody who doesn't get talked about in the Facebook story is Jeff Rothschild, who was an – the early adult supervision engineer, somebody exceeding worth listening to. There's a fantastic video presentation that he gave this, only available inside of Facebook that I insisted all my students watch, because it was just – No, this is concentrated essence of engineering right here and you're not going to get this anyplace else. He was attracted to this project.

I think there's a geek charisma to the way Zuck presented, so engineers would not roll their eyes. I think there's a knack to establishing a presence in a room full of engineers and he definitely had that space.

[0:45:23.6] JM: Not every startup has the growth pressures and the opportunity that Facebook has, but there are definitely engineering practices that the average company can adopt from Facebook. What are those engineering practices? What should the average startup, the average company take away from Facebook?

[0:45:56.9] KB: Yeah, my snap answer is nothing. People should figure out what their style is and do their style. I've been talking about software process for a long, long time. Something I notice is there are people who are uncomfortable taking responsibility. I'm one of those in my not so sane moments. They want to process where they can say, “Well, hey, we executed the process. We failed, but we executed the process.”

I think losing that and realizing that there's no such thing as a technical success, that you're all in it together and that your process is your process and you should play with it, you should experiment with it, you should try out a bunch of ideas. In the end, it needs to be yours. I think that's the real lesson.

Facebook did that. They did things that weren't conventional, not because they were unconventional, but because it made sense in the Facebook context. Everybody should be doing that and not copying – Spotify is the flavor of the month. Well, let's copy the Spotify model. Well, Spotify didn't copy the Spotify model, so what makes you think copying is the right thing to do?

That said, there are some intriguing engineering policies that I wouldn't have imagined would work, that ended up working quite well. Like this allocation process, where you got hired into the company and you chose your team from among the teams that had available headcount, except there's short tangent.

When I got there, teams would be over or understaffed compared to their headcount quite regularly. That was a good thing. The team I eventually ended up joining built in-person called culture, or infrastructure built the internal tools. Some managers were very happy that they were going to have a headcount allocation tool that would stop teams from poaching engineers.

This is ridiculous. Yes, it can't be random, but if 90% of the engineer, or if 90% of the teams have the number of people that we plan for them to have a year ago, that's got to be good enough, or maybe the right number is 80. If a 100% are adhering to decisions that were made on average six months ago, that can't be right. Yeah, that was an interesting conversation the day that came up. It's like, "Oh, we can enforce our policies." I'm like, "Maybe we shouldn't." Yeah, go ahead.

[0:49:28.7] JM: Given that you spent much of your career before Facebook and during your time at Facebook articulating software processes, and despite that, every pre-established notion that you had about software engineering was called into question in your first week at this new company, does it make you question what are we even doing writing books about software analysis and software practices and extreme programming? Can we really even say – or code complete.

Can we really even say anything concrete about design practices, or Kanban planning, or anything? What can we even be sure of?

[0:50:26.3] KB: You can only be fairly certain of the things you've tried yourself in your context, and only as – and those decisions are like fish. A month later, you should definitely question their value. If you let go of this low, what should our process be and say, what should our process be for refining our process, if you let go of the need for an answer and you embrace answering as a continuous process, then I don't think you can do better than that. That's part of my answer.

The other part of my is just because there isn't one way that works, there are a bunch of ways that work really badly. There are a few ways that work well, and I think of them as attractors in the space of process. There's a few ways that work well and there are a whole bunch of ways that work horribly. The first thing to do is identify where you're doing something that's horrible and stop doing that.

Yes, I think we can write useful stories about software development process, but they're stories, they're not recipes. The person listening to the story is going to have to take it in and digest it and apply it in their own specific context, because every single day at every single company is a different context. Of course, the answer is going to be different. The inputs are different.

[0:52:14.9] JM: Last question, what do you miss most about working at Facebook?

[0:52:19.1] KB: Scale. Just that leverage, that first feature that I shipped was a civil union and domestic partnership became relationship types. You could say, "I am in a domestic partnership with such and so." Last time I checked, that was in use by a million and a half people in their profile. That ability to have a huge impact on the world was intoxicating.

Another boot camp project, I looked at the photos code and I thought, "Something's a little wrong with the data fetching. I can make this more efficient." I accidentally saved 5 million dollars a year. Like, whoa. As an engineer, being able to have that leverage is just amazing.

At the same time, watching the troubles that Facebook has gone through reminds me that explore and expand are not the same as extract. When you get to extract, there's a big downside and you have to stop only looking at possible improvements and metrics, or however you're going to measure success. You have to start paying attention to the downsides of your actions in a way that you don't when you're little and scrappy you got nothing to lose.

I think that's the challenge that Facebook faces.

Honestly, if I had to put money on it, I don't – because everybody is so focused on impact and impact is this magic word at Facebook and if you don't have impact, you're fired. If you do have impact, you're rewarded. Everybody's looking for that upside. Those upsides are getting smaller and smaller, but the downsides are getting bigger and bigger. Culturally, how do you unwind that? I would guess that they won't be able to.

[0:54:23.0] JM: Kent Beck, thanks for coming on the show. It's been really fun talking to you.

[0:54:26.0] KB: My pleasure.

[0:54:27.1] JM: Okay. Great show. If you don't mind, stop your client side backup record –

[END]